A taxonomy of market-based resource management systems for utility-driven cluster computing

Chee Shin Yeo and Rajkumar Buyya
Grid Computing and Distributed Systems Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne
VIC 3010, Australia
Email: {csyeo, raj}@cs.mu.oz.au

December 8, 2004

Abstract

In utility-driven cluster computing, cluster systems need to know the specific needs of different users so as to allocate resources according to their needs. They are also vital in supporting service-oriented Grid computing that harness resources distributed worldwide based on users' objectives. Market-based resource management systems make use of real-world market concepts and behavior to assign resources to users. This paper outlines a taxonomy that describes how market-based resource management systems can support utility-driven cluster computing. The taxonomy is used to survey existing market-based resource management systems to better understand how they can be utilized.

I. Introduction

The next-generation scientific research involves solving Grand Challenge Applications (GCAs) that demand ever increasing amount of computing power. Recently, a new type of High Performance Computing (HPC) paradigm called *cluster computing* [1][2][3] has become a more viable choice for executing these GCAs since cluster systems are able to offer equally high performance with a lower price compared to traditional supercomputing systems. A cluster system comprises of independent machines that are connected by high-speed networks and uses middlewares that create an illusion of single system [4] and hide the complexities of underlying cluster architecture from the users. For example, the *cluster Resource Management System (RMS)* is a middleware that manages the resources and seamless execution of jobs in a cluster of computers.

Existing cluster RMSs still adopt system-centric resource allocation approaches that maximizes overall job performance and system usage. These system-centric approaches assume that all job requests are of equal importance and thus neglect actual levels of service required by different users. *Market-based RMSs* [5][6][7][8][9] have a greater emphasis on user QoS requirements as opposed to traditional RMSs that focus on maximizing system usage. Market concepts can be used to prioritize competing jobs and assign resources to jobs according to users' valuation for QoS requirements and cluster resources.

Market-based cluster RMSs need to support three requirements in order to enable *utility-driven cluster computing* [7]: (i) provide means for users to specify their Quality of Service (QoS) needs and valuations, (ii) utilize policies to translate the valuations into resource allocations, and (iii) support mechanisms to enforce the resource allocations in order to achieve each individual user's perceived value or utility. The first requirement allows the market-based cluster RMS to be aware of user-centric service requirements so that competing service requests can be prioritized more accurately. The second requirement then determines how the cluster RMS can allocate resources appropriately and effectively to different requests by considering

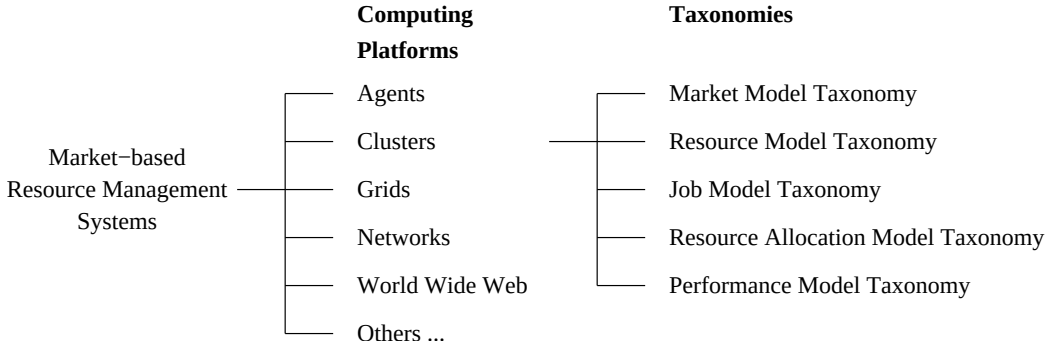

Fig. 1. Categorization of market-based resource management systems.

the solicited service requirements. The third requirement finally needs the underlying cluster operating system mechanisms to recognize and enforce the assigned resource allocations.

The advent of Grid computing [10] further reinforces the necessity for utility-driven cluster computing. In service-oriented Grid computing [11], users can submit jobs with specific QoS requirements to Grid schedulers such as Grid brokers [12][13] and Grid workflow engines [14] that discover suitable Grid resources to execute their jobs. Since most Grid resources are cluster systems, the cluster RMS needs to support service level agreement based resource allocations. This means that the cluster RMS has to not only balance competing user needs, but also enhance the profitability of the cluster owner while delivering the expected level of service performance. In addition, market concepts and mechanisms incorporated at the cluster computing level facilitates easy extensions to support Grid economy [15] and enforce service level agreements in service-oriented Grids.

Market-based RMSs have been utilized in many different computing platforms: agents [16][17][18], clusters [19][7][9], Grids [8][20][21], networks [22][23][24][25] and world wide web [26][27][28] (see Fig. 1). In this paper, we focus on developing a taxonomy that classifies market-based RMSs in the context of utility-driven cluster computing. The taxonomy consists of five sub-taxonomies, namely *market model*, *resource model*, *job model*, *resource allocation model*, and *performance model* (see Fig. 1). Researchers can use this taxonomy to gain a better understanding of key design factors and issues that are crucial in developing effective market-based cluster RMSs to support utility-driven cluster environment. We also present an abstract model to conceptualize the essential functions of a market-based cluster RMS and include a survey to demonstrate how the taxonomy can be applied.

II. Related Work

There are several proposed taxonomies for scheduling in distributed and heterogeneous computing. However, none of these taxonomies focus on market-based cluster computing environments. The taxonomy in [29] classifies scheduling strategies for general-purpose distributed systems. In [30], two taxonomies for state estimation and decision making are proposed to characterize dynamic scheduling for distributed systems. The EM^3 taxonomy in [31] utilizes the number of different execution modes and machine models to identify and classify heterogeneous systems. In [32], a modified version of the scheduling taxonomy in [31] is proposed to describe the resource allocation of heterogeneous systems. The taxonomy in [33] considers three characteristics of heterogeneous systems: application model, platform model and mapping strategy to define resource matching and scheduling. A taxonomy on Grid resource management system [34] includes a scheduling sub-taxonomy that examines four scheduling characteristics: scheduler organization, state estimation, rescheduling and scheduling policy. But, our taxonomy focuses on market-based RMSs for utility-driven cluster computing where cluster systems have a number of significant differences compared to Grid systems. One key difference is that a cluster system is distributed within a single administrative domain, whereas a Grid system is distributed across multiple administrative domains.

III. Definitions and Requirements for Utility-driven Cluster Computing

In cluster computing, the *producer* is the owner of the cluster system that provides resources to accomplish users' service requests. Examples of resources that can be utilized in a cluster system are processor power, memory storage and data storage. The *consumer* is the user of the resources provided by the cluster system and can be either a physical human user or a software agent that represents a human user and acts on his behalf. A cluster system has multiple consumers submitting job requests that need to be executed.

The *cluster RMS* creates the Single System Image (SSI) [4] for a cluster system by providing a uniform interface for user-level sequential and parallel applications to be executed on the cluster system to hide the existence of multiple cluster nodes from users. It supports four main functionalities: resource management, job queuing, job scheduling, and job execution. The cluster RMS manages and maintains status information of the resources such as processors and disk storage in the cluster. Jobs submitted into the cluster system are initially placed into queues until there are available resources to execute the jobs. The cluster RMS then invokes a scheduler to determine how resources are assigned to jobs. After that, the cluster RMS dispatches the jobs to the assigned nodes and manages the job execution processes before returning the results to the users upon job completion.

Existing cluster RMSs such as Condor [35], LoadLeveler [36], Load Sharing Facility (LSF) [37], Portable Batch System (PBS) [38], and Sun Grid Engine (SGE) [39] are not viable to support utility-driven cluster computing since they still adopt system-centric resource allocation approaches that focus on optimizing overall cluster performance. For example, these cluster RMSs aim to maximize processor throughput and utilization for the cluster, and minimize average waiting time and response time for the jobs. They neglect the need to use utility models for allocation and management of resources that would otherwise consider and thus achieve the desired utility for cluster users and owners. Therefore, these existing cluster RMSs need to be extended to support utility-driven cluster computing.

In *utility-driven cluster computing*, consumers have different requirements and needs for various jobs and thus can assign value or utility to their job requests. During job submission to the cluster RMS, consumers can specify their requirements and preferences for each respective job using QoS parameters. The cluster RMS then considers these QoS parameters when

making resource allocation decisions. This provides a user-centric approach with better user personalization since consumers can potentially affect the resource allocation outcomes, based on their assigned utility.

However, the producer has the final control over the resource allocation decision since he owns the cluster system and thus implements the resource allocation policies. Depending on his objective, the producer may want to maximize utility for himself or the consumers. For instance, a producer wants to maximize overall social welfare and consumers' utility satisfaction. The cluster system can probably achieve this objective from either the job perspective where it maximizes the number of jobs whose QoS is satisfied or the consumer perspective where it maximize the aggregate utility perceived by individual consumers. On the other hand, the producer may want to maximize his own personal benefit, such as maximizing monetary profits when consumers provide different monetary offers for satisfying their job requests.

Next-generation service-oriented Grid computing allows Grid users to specify various level of service required for processing their jobs on a Grid. Grid schedulers then make use of this user-specific information to discover available Grid resources and determine the most suitable Grid resource to submit the jobs to. Currently, cluster systems dominate the majority of Grid resources whereby Grid schedulers can submit and monitor their jobs being executed on the cluster systems through interaction with their cluster RMS. Examples of large-scale Grid systems that are composed of cluster systems includes the TeraGrid [40] in United States, LHC Computing Grid [41] in Europe, NAREGI [42] in Japan, and APAC Grid [43] in Australia.

In addition, commercial vendors are progressing aggressively towards providing a service market through Grid computing. For instance, IBM's E-Business On Demand [44], HP's Adaptive Enterprise [45] and Sun Microsystem's pay-as-you-go [46] are using Grid technologies to provide dynamic service delivery where users only pay for what they use and thus save from investing heavily on computing facilities. Vendors and respective users have to agree on service level agreements that serve as contracts outlining the expected level of service performance such that vendors are liable to compensate users for any service under-performance. This further reinforces the significance of using market-based mechanisms to enable utility-driven cluster computing so that user-specific service requests across service-oriented Grids can be fulfilled successfully to enforce service level agreements.

IV. Abstract Model for Market-based Cluster Resource Management System

Fig. 2 outlines an abstract model for the market-based cluster RMS. The purpose of the abstract model is to identify generic components that are fundamental and essential in a market-based cluster RMS and portray the interactions between these components. Thus, the abstract model can be used to study how existing cluster RMS architectures can be leveraged and extended to incorporate market-based mechanisms to support utility-driven cluster computing.

The market-based cluster RMS consists of two primary entities: cluster manager and cluster node. For implementations within cluster systems, the machine that operates as the cluster manager can be known as the manager, server or master node and the machine that operates as the cluster node can be known as the worker or execution node. The actual number of cluster manager and cluster nodes depends on the implemented management control. For instance, a simple and common configuration for cluster systems is to support centralized management control where a single cluster manager collates multiple cluster nodes into a pool of resources as shown in Fig. 2.

The cluster manager serves as the front-end for users and provides the scheduling engine responsible for allocating cluster resources to user applications. Thus, it supports two interfaces: the manager-consumer interface to accept requests from consumers and the manager-worker interface to execute requests on selected cluster nodes. The consumers can be actual user applications, service brokers that act on the behalf of user applications or other cluster RMSs such as those operating in multi-clustering or Grid federation environments where requests that cannot be fulfilled locally are forwarded to other cooperative clusters.

When a service request is first submitted, the request examiner interprets the submitted request for QoS requirements such as deadline and budget. The admission control then determines whether to accept or reject the request in order to ensure that the cluster system is not overloaded where many requests will not be fulfilled successfully. The scheduler selects suitable worker nodes to satisfy the request and the dispatcher starts the execution on the selected worker nodes. The node status/load monitor keeps track of the availability of the nodes and their workload, while the job monitor maintains the execution progress of requests.

It is vital for a market-based cluster RMS to support pricing and accounting mechanisms. The pricing mechanism decides how requests are charged. For instance, requests can be charged based on submission time (peak/off-peak), pricing rates (fixed/changing) or availability of resources (supply/demand). Pricing serves as a basis for managing the supply and demand of cluster resources and facilitates in prioritizing resource allocations effectively. The accounting mechanism maintains the actual usage of resources by requests so that the final cost can be computed and charged to the consumers. In addition, the maintained historical usage information can be utilized by the scheduler to improve resource allocation decisions.

The cluster nodes provide the resources for the cluster system to execute service requests via the worker-manager interface. The job control ensures that requests are fulfilled by monitoring execution progress and enforcing resource assignment for executing requests.

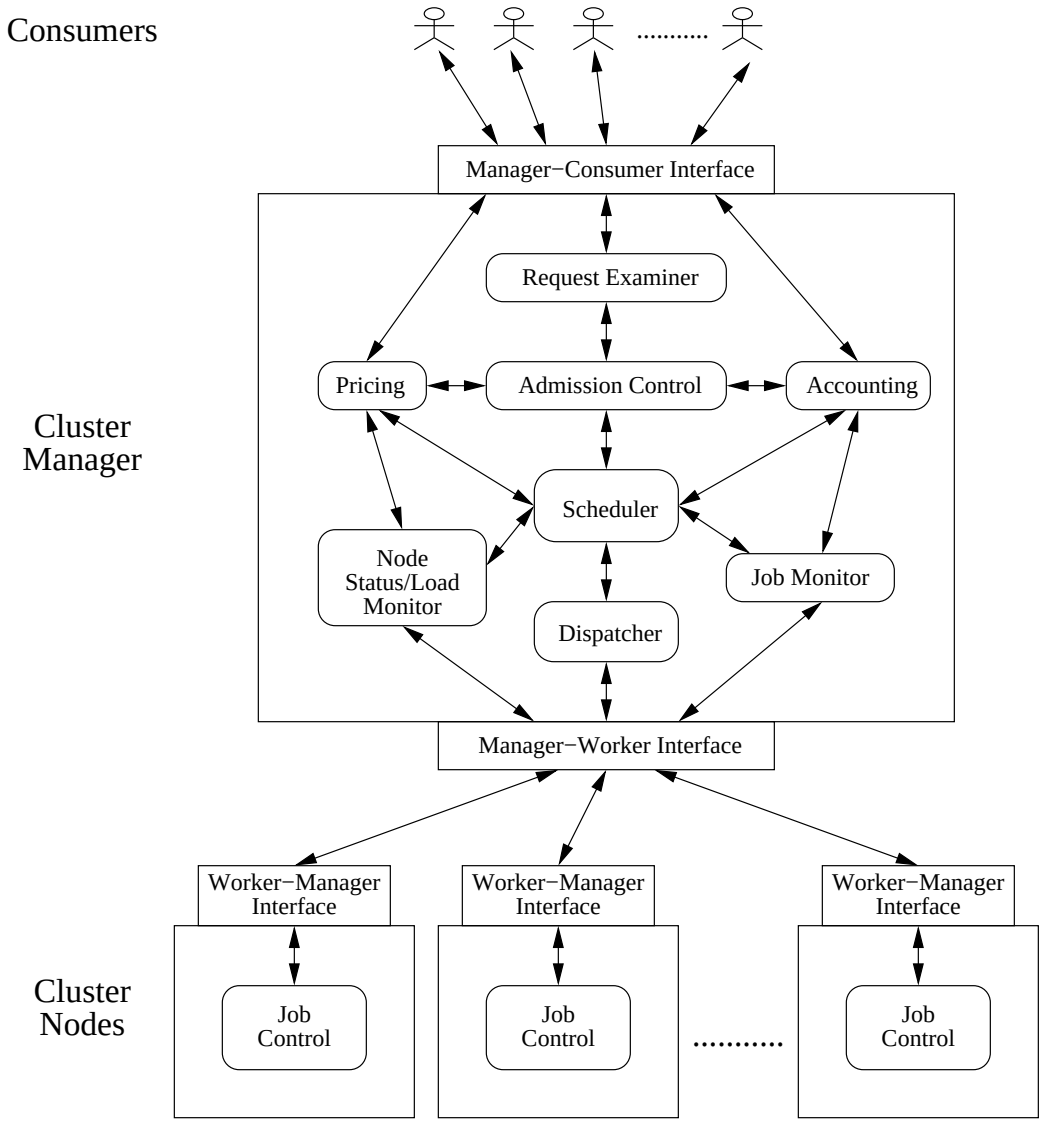

Fig. 2. Abstract Model for market-based cluster RMS.

V. TAXONOMY

The taxonomy classifies market-based RMSs based on various perspectives in order to identify key factors and issues relevant to the context of utility-driven cluster computing.

A. Market Model Taxonomy

The *market model* taxonomy examines how market concepts present in real-world human economies are incorporated into market-based RMSs. This allows developers to understand what market-related attributes need to be considered, in particular to deliver utility. The market model taxonomy comprises of four sub-taxonomies: economic model, participant focus, trading environment, and QoS attributes (see Fig. 3).

1) Economic Model: The *economic model* derived from [47] establishes how resources are allocated in a market-driven computing environment. Selection of a suitable economic model primarily depends on the market interaction required between the consumers and producers.

For *commodity market*, producers specify prices and consumers pay for the amount of resources they consume. Pricing of resources can be determined using various parameters, such as usage time and usage quantity. There can be flat or variant pricing rates. A flat rate means that pricing is fixed for a certain time period, whereas, a variant rate means that pricing changes over time, often based on the current supply and demand at that point of time. A higher demand results in a higher variant rate.

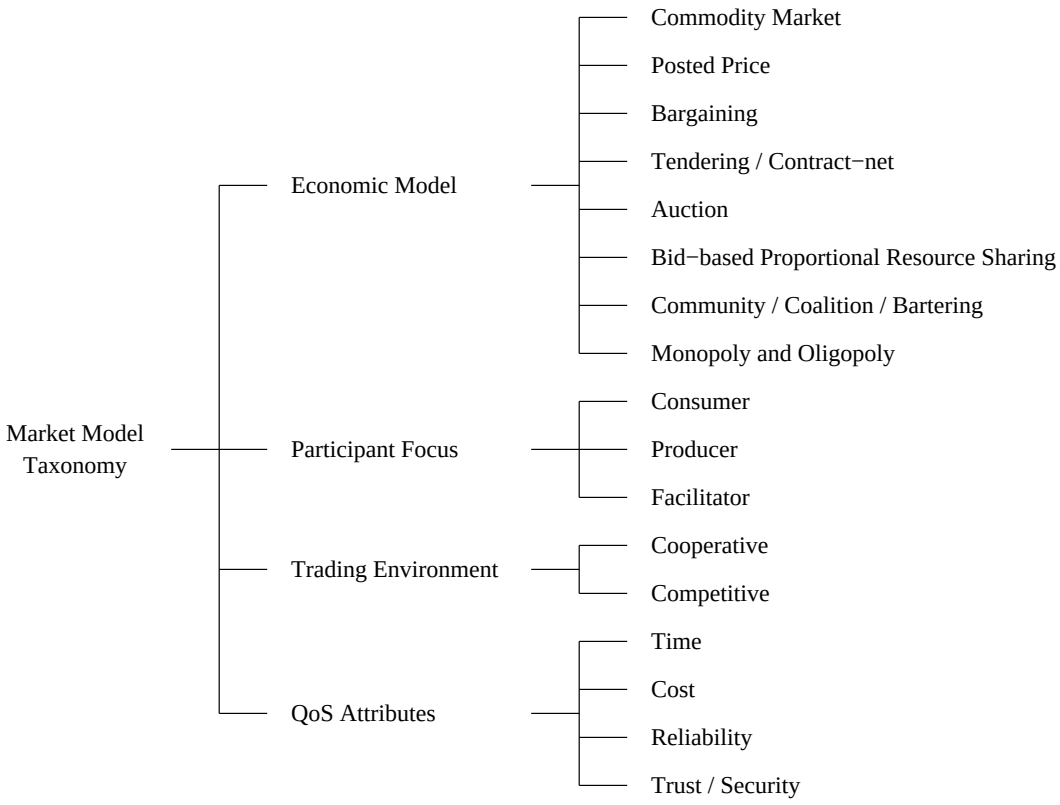

Fig. 3. Market model taxonomy.

Posted price operates similarly as the commodity market. However, special offers are advertised openly so that consumers are aware of discounted prices and can thus utilize the offers. *Bargaining* enables both producers and consumers to negotiate for a mutually agreeable price. Producers typically start with higher prices to maximize profits, but consumers start with lower prices to minimize costs. Negotiation stops when the producer or consumer does not wish to negotiate further or a mutually agreeable price has been reached. Bargaining is often used when supply and demand prices cannot be easily defined.

In *tendering/contract-net*, the consumer first announces its requirements to invite bids from potential producers. Producers then evaluate the requirements and can respond with bids if they are interested and capable of the service or ignore the announcement if they are not interested or too busy. The consumer consolidates bids from potential producers, select the most suitable producer, and send a tender to the selected producer. The tender serves as a contract and specifies conditions that the producer has to accept and conform to. Penalties may be imposed on producers if the conditions are not met. The selected producer accepts the tender and delivers the required service. The consumer then notifies other producers of the unsuccessful outcome. Tendering/contract-net allows a consumer to locate the most suitable producer to meet its service request. However, it does not always guarantee locating the best producer each time since potential producers can choose not to respond or are too busy.

Auction allows multiple consumers to negotiate with a single producer by submitting bids through an auctioneer. The auctioneer acts as the coordinator and sets the rules of the auction. Negotiation continues until a single clearing price is reached and accepted or rejected by the producer. Thus, auction regulates supply and demand based on the number of bidders, bidding price and offer price. There are basically five primary types of auctions, namely english, first-price, vickrey, dutch and double [47].

Bid-based proportional resource sharing assigns resources proportionally based on the bids given by the consumers. So, each consumer will be allocated a proportion of the resources as compared to a typical auction model where only one consumer with the winning bid is entitled to the resource. This is ideal for managing a large shared resource where multiple consumers are equally entitled to the resource. *Community/coalition/bartering* supports a group of community producers/consumers who shares each others' resources to create a cooperative sharing environment. This model is typically adopted in computing environments where consumers are also producers and thus both contribute and use resources. Mechanisms are required to regulate that participants act equally in both roles of producers and consumers for fairness. *Monopoly/oligopoly* depicts a non-competitive market where only a single producer (monopoly) or a number of producers (oligopoly) determines the market

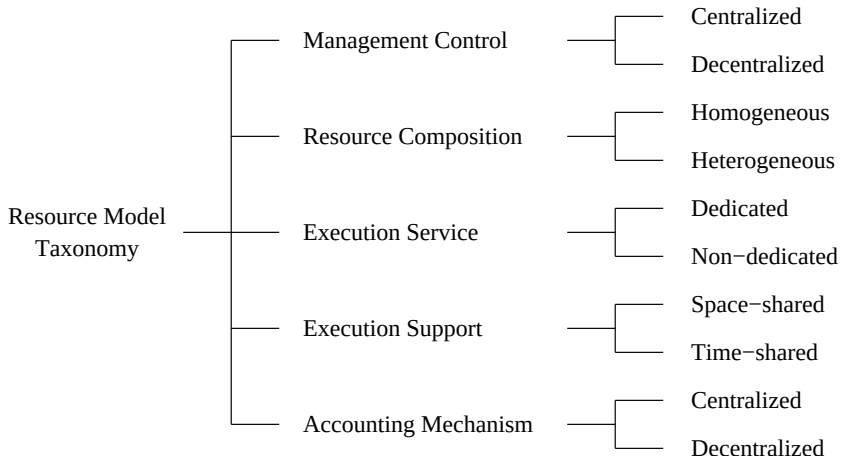

Fig. 4. Resource model taxonomy.

price. Consumers are not able to negotiate or affect the stated price from the producers.

Some market-based RMSs may use hybrids or adopt modified variants of the above mentioned economic models in order to harness the strengths of different models and provide improved customizations based on user-specific application criteria.

2) Participant Focus: The *participant focus* identifies the party for whom the market-based RMS aims to achieve benefit or utility. Having a *consumer* participant focus implies that a market-based RMS aims to meet the requirements specified by cluster users, and possibly optimize their perceived utility. For instance, the consumer may want to spend minimal budget for a particular job. Similarly, a *producer* participant focus results in resource owners fulfilling their desired utility. It is also possible to have a *facilitator* participant focus whereby the facilitator acts like an exchange and gains profit by coordinating and negotiating resource allocations between consumers and producers.

Utility-driven cluster computing should focus primarily on achieving utility for the consumers as the key purpose of the cluster systems is to satisfy end-users' demand for resources to execute their supercomputing applications. However, producers and facilitators may have specific requirements that also need to be taken into consideration and not neglected totally.

3) Trading Environment: The *trading environment* generalizes the motive of trading between the parties or participants that is supported via the market-based RMS. It is established based on the needs and aims of various parties and will determine the trading relationships between them. A *cooperative* trading environment promotes collective sharing of resources where producers may create a federation of resources to speed up execution of jobs. On the other hand, a *competitive* trading environment supports individualistic resource usage where consumers have to contend with other consumers to obtain any available resources. A market-based RMS can only support either a cooperative or competitive trading environment, but not both.

4) QoS Attributes: QoS attributes describe service requirements which consumers require the producer to provide in a service market. The *time* QoS attribute identifies the time required for various operations. Some examples of time QoS attributes are job execution time, data transfer time and deadline required by the consumer for the job to be completed. The *cost* QoS attribute depicts the cost involved for satisfying the job request of the consumer. A cost QoS attribute can be monetary such as the budget that a consumer is willing to pay for the job to be completed, or non-monetary in other measurement units such as the data storage (in bytes) required for the job to be executed.

The *reliability* QoS attribute represents the level of service guarantee that is expected by the consumer. For instance, jobs that require high reliability will require the market-based cluster RMS to be highly fault-tolerant whereby check-pointing and backup facilities, with fast recovery after service failure are incorporated. The *trust/security* QoS attribute determines the level of security needed for executing applications on resources. For example, jobs with highly sensitive and confidential information will require a resource with high trust/security to process.

Market-based RMSs should support time, cost and reliability QoS attributes since they are critical in enabling a service market for utility-driven cluster computing. The trust/security QoS attribute is also critical if the user applications require secure access to resources. Satisfying QoS attributes is highly critical in a service market as consumers pay based on the different levels of service required. The market-based RMS should be able to manage service demands without sacrificing existing service requests and resulting in service failures. Failure to meet QoS attributes will only require the producer to compensate consumers, but also have a bad reputation on the producer that affects future credibility.

B. Resource Model Taxonomy

The *resource model* taxonomy describes architectural characteristics of cluster systems. It is important to design market-based resource management systems that conform to the clusters' underlying system architectures and operating environments since there may be certain cluster system attributes that can be exploited. The resource model taxonomy comprises of five sub-taxonomies: management control, resource composition, execution service, execution support, and accounting mechanism (see Fig. 4).

1) Management Control: The *management control* depicts how the resources are managed and controlled in the cluster systems. A cluster with *centralized* management control has a single centralized resource manager that administers all the resources and jobs in the cluster. On the contrary, a cluster with *decentralized* management control has more than one decentralized resource managers managing subsets of resources within a cluster. Decentralized resource managers need to communicate with one another in order to be informed of local information of other managers.

A centralized resource manager collects and stores all local resource and job information within the cluster at a single location. Since a centralized resource manager has the global knowledge of the entire state of the cluster system, it is easier and faster for a market-based RMS to communicate and coordinate with a centralized resource manager, as opposed to several decentralized resource managers. A centralized resource manager also allows a large change to be incorporated in the cluster environment as the change needs to be updated at a single location only. However, a centralized management control is more susceptible to bottlenecks and failures due to the overloading and malfunction of the single resource manager. A simple solution is to have backup resource managers that can be activated when the current centralized resource manager fails. In addition, centralized control architectures are less scalable compared to decentralized control architectures. Since centralized and decentralized management have various strengths and weaknesses, they perform better for different environments.

Centralized management control is mostly implemented in cluster systems since they are often owned by a single organization and modeled as a single unified resource. Therefore, a market-based RMS should be designed to support centralized management control, but may also support decentralized management control for portability reasons.

2) Resource Composition: The *resource composition* defines the combination of resources that make up the cluster system. A cluster system with *homogeneous* resource composition consists of all worker nodes having the same resource components and configurations, whereas *heterogeneous* resource composition consists of worker nodes having different resource components and configurations. A homogeneous resource composition enables faster and more efficient execution, while a heterogeneous resource composition can support execution of distinct applications based on different resource demands. Thus, it may be beneficial for cluster systems with heterogeneous resource composition to have different sets of worker nodes with homogeneous resource composition within each set.

Market-based RMSs should be able to function uniformly for both homogeneous and heterogeneous resource compositions. For heterogeneous resource composition, market-based RMSs must have effective means of translating user-defined requirements to equivalent measures on each specific execution node to ensure accuracy of QoS requirements. In addition, market-based RMSs also need to translate load measures correctly across heterogeneous cluster nodes to support load balancing.

3) Execution Service: The *execution service* reflects the service availability of the cluster system. A cluster system with *dedicated* execution service enables users to have full dedicated access at all times and submitted jobs can be executed instantly if there are free resources available. On the other hand, a cluster system with *non-dedicated* execution service is not always available for users to submit and execute jobs. A cluster system may consist of separate sets of worker nodes that provide dedicated and non-dedicated execution services. It is possible for cluster systems to have execution service that is dynamic and switches between dedicated and non-dedicated modes alternately. Examples of RMSs that support such service are Condor [35] and Alchemi [48].

Market-based RMSs should be able to operate in both dedicated and non-dedicated execution services. It will be ideal to design market-based RMSs that can automatically detect and self-adapt to changing execution services.

4) Execution Support: The *execution support* determines the type of processing that is supported by the cluster's underlying operating system. The *space-shared* execution support enables only a single job to be executed at any one time on a processor, whereas the *time-shared* execution support allows multiple jobs to be executed at any one time on a processor.

This implies that time-shared execution support can lead to a higher throughput of jobs over a period of time. However, space-shared execution support finishes a job faster as a single job is executed using the full processing power of the processor. In addition, time-shared execution support allows idle processing time to be reallocated to other jobs if a job is not using the allocated processing power such as when reading or writing data. On the other hand, such preemption is not permitted in space-shared execution support since the processor is dedicated to a single job only, thus wasting the unused processing time. Market-based RMSs should provide both space-shared and time-shared execution supports.

5) Accounting mechanism: The *accounting mechanism* maintains and stores information about job executions in the cluster system. The stored accounting information may then be used for charging purposes or planning future resource allocation decisions. A *centralized* accounting mechanism denotes that information for the entire cluster system is maintained by a single centralized accounting manager and stored on a single node. A *decentralized* accounting mechanism indicates that multiple

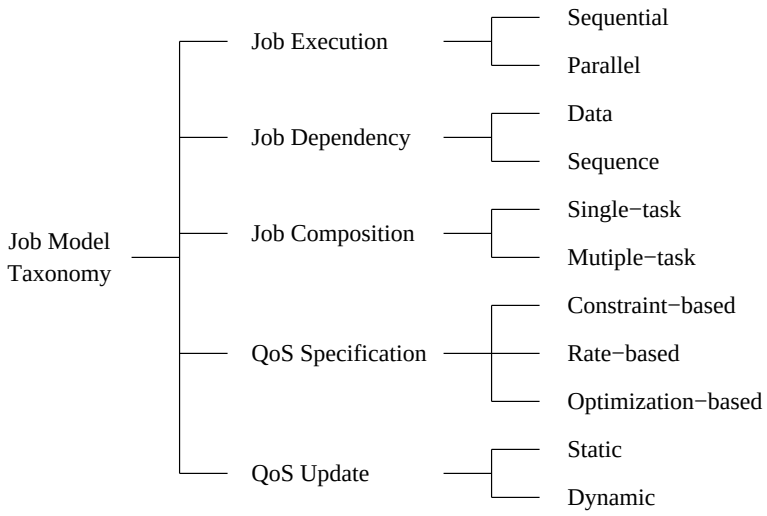

Fig. 5. Job model taxonomy.

decentralized accounting managers monitor and store separate sets of information on multiple nodes. Examples of accounting mechanisms that supports charging are GridBank [49] and QBank [50].

In GridBank, each Grid resource uses a Grid Resource Meter to monitor the usage information and a GridBank Charging Module to compute the cost. The centralized GridBank server then transfers the payment from the users' bank accounts to the Grid resource's account. On the other hand, QBank supports both centralized and decentralized configurations. For instance, the simplest and most tightly-coupled centralized QBank configuration is having a central scheduler, bank server and database for all resources which is suitable for a cluster environment. QBank also allows multiple schedulers, bank servers and databases for each separate resource in different administrative domains to support a highly decentralized P2P or Grid environment.

Similar to the management control taxonomy, it is easier for market-based RMSs to access information based on the centralized accounting mechanism. But, the centralized accounting mechanism is less reliable and scalable compared to the decentralized accounting mechanism. Market-based RMSs should be designed to support both centralized and decentralized accounting mechanisms.

C. Job Model Taxonomy

The *job model* taxonomy categorizes attributes of jobs that are to be executed on the cluster systems. Market-based RMSs need to take into account of job attributes to ensure that different job types with distinct requirements can be fulfilled successfully. The job model taxonomy comprises of five sub-taxonomies: job execution, job dependency, job composition, QoS specification, and QoS update (see Fig. 5).

1) Job Execution: The *job execution* describes the processing that is required by the job. Market-based RMSs can then determine how to assign suitable nodes that are able to support the type of processing required.

For *sequential* job execution, the job executes on one processor independently. For *parallel* job execution, the parallel job has to be distributed to multiple processors before executing these multiple processes simultaneously. Thus, parallel job execution speeds up processing and is often used for solving complex problems. One common type of parallel job execution is called message-passing where multiple processes of a parallel program on different processors interact with one another via sending and receiving messages.

Market-based RMSs should support both sequential and parallel job executions. When supporting parallel job execution, market-based RMSs need to determine that the required number of processors is available before executing the job, which is more complex compared to supporting sequential job executions.

2) Job Dependency: The *job dependency* identifies any dependencies that jobs have to rely on before they can be executed. The *data* job dependency has jobs requiring input data in order to be executed. These input data may be available at remote locations and need to be transferred to the execution node or are not available yet as some jobs waiting to be executed will generate the data. The *sequence* job dependency has jobs that need to be executed in a pre-defined order. For instance, the job which performs initiations needs to be executed first before other jobs can be executed. Currently, only higher-level schedulers such as Condor DAGMan [35] and Gridbus workflow engine [14] support execution of dependent jobs expressed in Directed Acyclic Graphs (DAG).

Market-based RMSs need to consider both data and sequence job dependencies, so that overall user requirements for a set of dependent jobs can still be met successfully. There is a need to prioritize dependencies between dependent jobs. For

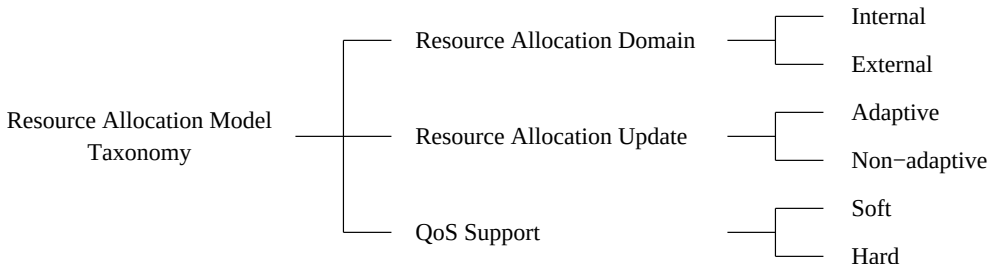

Fig. 6. Resource allocation model taxonomy.

example, a parent job with more dependent child jobs has a higher priority than a parent job with fewer child jobs. To speed up processing, it is possible to execute independent sets of dependable jobs in parallel since jobs are only dependent on one another within a set and not across sets.

3) Job Composition: The *job composition* portrays the collection of tasks within a job that is defined by the user. The *single-task* job composition refers to a job having a single task, while the *multiple-task* job composition refers to a job being composed of multiple tasks. For instance, a parameter sweep job is composed of multiple independent tasks, each with a different parameter so that the multiple tasks can be executed in parallel to reduce the overall processing time of the parameter sweep job.

Market-based RMSs should able to support both single-task and multiple-task job compositions. With multiple-task job composition, the market-based RMS needs to schedule and monitor tasks to ensure that the overall job requirements will be satisfied. It may be possible that a multiple-task job composition has task dependencies similar to job dependencies described in the job dependency sub-taxonomy described above, but task dependencies are restricted to within a job as opposed to job dependencies that span across multiple jobs.

4) QoS Specification: The *QoS specification* describes how users can specify their QoS requirements for a job to indicate their perceived level of utility. This provides cluster users with the capability to influence the resource allocation outcome in the cluster.

Users can define *constraint-based* QoS specifications that use bounded value or range of values for a particular QoS so that the minimal QoS requirements can be fulfilled. Some examples of constraint-based QoS specifications that users can specify are execution deadline, execution budget, memory storage size, disk storage size and processor power. For instance, a user can specify a deadline less than one hour (value) or deadline between one and two hours (range of values) for executing a job on cluster nodes with available memory storage size of more than 256 MB (value) and processor speed between 200 GHz and 400 GHz (range of values).

A *rate-based* QoS specification allows users to define constant or variable rates that signify the required level of service over time. For instance, a user can specify a constant cost depreciation rate of ten credits per minute such that the user pays less for a slower job completion time. To support a higher level of personalization, users can state *optimization-based* QoS specifications that identify specific QoS to optimize in order to maximize the users' utility. An example is a user wants to optimize the deadline of his job so that the job can be completed in the shortest possible time.

Market-based RMSs should at least provide either constraint-based or rate-based QoS specification so that the required utility is considered when making resource assignment decisions and thus satisfied. It will be ideal to support optimization-based QoS specifications so that the best combined optimal outcome can be achieved for various users.

5) QoS Update: The *QoS update* determines whether QoS requirements of jobs can change after jobs are submitted and accepted. The *static* QoS update means that the QoS requirements are fixed and do not change once the job is submitted, while the *dynamic* QoS update means that QoS requirements of the jobs can change. These dynamic changes may already be pre-defined during job submission or modified by the user during an interactive job submission session.

Market-based RMSs need to support both static and dynamic QoS updates. For dynamic QoS update, the market-based RMS needs to reassess the new changed QoS requirements and revise the resource assignments accordingly. This is because the new QoS requirements may result in previous resource assignments being void and unable to meet the new requirements. In addition, it is highly probable that other planned or executing jobs may also be affected so there is a need to reassess and reallocate resources to minimize any possible adverse effects.

D. Resource Allocation Model Taxonomy

The *resource allocation model* taxonomy analyzes factors that can influence how the market-based RMS operates and thus resource assignment outcome. The resource allocation model taxonomy comprises of three sub-taxonomies: resource allocation domain, resource allocation update and QoS support (see Fig. 6).

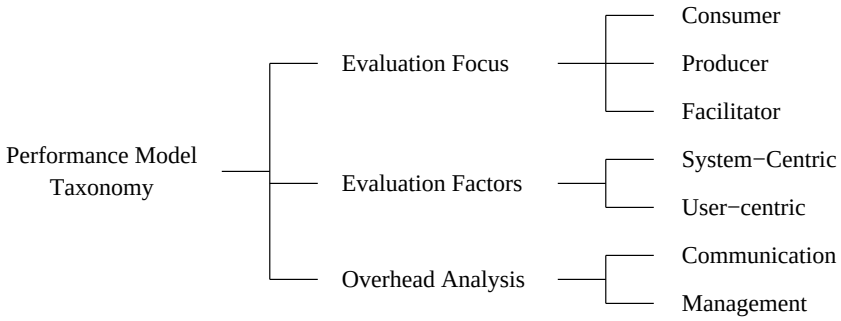

Fig. 7. Performance model taxonomy.

1) Resource Allocation Domain: The *resource allocation domain* defines the scope that the market-based RMS is able to operate in. Having an *internal* resource allocation domain restricts the assignment of jobs to within the cluster system. An *external* resource allocation domain allows the market-based RMS to assign jobs externally outside the cluster system, meaning that jobs can be executed on other remote cluster systems. Remote cluster systems may be in the same administrative domain belonging to the same producer such as an organization which owns several cluster systems or in different administrative domain owned by other producers such as several organizations which individually own some cluster systems. For instance, Sun Grid Engine (SGE) [39] enables a cluster system to allocate jobs externally to other cluster systems within the same administrative domain by using tickets to control the quota of jobs that users can submit.

Thus, supporting external resource allocation domain allows the market-based RMS to have access to more alternative resources to satisfy more user requests. But, there is a need to address other issues such as data transfer times, network delays and reliability of remote cluster systems. For more flexibility and scalability, market-based RMSs should support both internal and external resource allocation domains.

2) Resource Allocation Update: The *resource allocation* update identifies whether the market-based RMS is able to detect and adapt to changes to maintain effective scheduling. Some examples of changes that can occur include availability of resources, amount of submission workload and varying job requirements. The *adaptive* resource allocation update means that the market-based RMS is able to adjust dynamically and automatically to suit any new changes. On the other hand, the *non-adaptive* resource allocation update means that it is not able to adapt to changes and thus still proceed on with its original resource assignment decision. Market-based RMSs should support adaptive resource allocation update since there is a possibility of improving a previous resource assignment decision based on the changed operating scenario.

3) QoS Support: The *QoS support* derived from [34] determines whether QoS specified by the user can be achieved. Admission control is essential during job submission to determine and feedback to the user whether the requested QoS can be given. If accepted by the admission control, the job needs to be monitored to ensure that the required QoS is enforced and fulfilled.

The *soft* QoS support allows user to specify QoS parameters, but do not guarantee that these service requests can be satisfied. On the contrary, the *hard* QoS support is able to ensure that the specified service will definitely be achieved. To support utility-driven cluster computing, market-based RMSs need to provide hard QoS support. This is non-trivial as a high degree of coordination and monitoring is required to enforce the QoS.

E. Performance Model Taxonomy

The *performance model* taxonomy outlines how market-based RMSs need be evaluated to measure their effectiveness and efficiency for supporting utility-driven cluster computing. The performance model taxonomy comprises of three sub-taxonomies: evaluation focus, evaluation factors and overhead analysis (see Fig. 7).

1) Evaluation Focus: The *evaluation focus* identifies the party that the market-based RMS is supposed to achieve utility for. The evaluation focus is similar to the participant focus sub-taxonomy discussed previously since it is obvious to measure performance based on the selected participant focus.

The *consumer* evaluation focus measures the level of utility that has been delivered to the consumer based on its requirements. Likewise, the *producer* and *facilitator* evaluation focus evaluates how much value is gained by the producer and facilitator respectively. For example, Libra [51] evaluates the utility achieved for consumers (users) via the Job QoS Satisfaction metric and the benefits gained by the producer (cluster owner) via the Cluster Profitability metric. Market-based RMSs should be able to achieve the required utility for the selected participant focus.

2) Evaluation Factors: Evaluation factors are metrics defined to determine the effectiveness of different market-based RMSs in providing utility-driven cluster computing. *System-centric* evaluation factors measure performance from the system perspective and thus depict the overall operational performance of the cluster system. Examples of system-centric evaluation

factors are average waiting time, average response time, system throughput, and resource utilization. Average waiting time is the average time that a job has to wait before commencing execution, while average response time is the average time taken for a job to be completed. System throughput determines the amount of work completed in a period of time, whereas resource utilization reflects the usage level of the cluster system.

User-centric evaluation factors assess performance from the participant perspective and thus portray the utility achieved by the participants. Different user-centric evaluation factors can be defined for assessing different participants that include consumer, producer or facilitator (as defined in the evaluation focus sub-taxonomy). For instance, Libra defines the Job QoS Satisfaction evaluation factor for consumer evaluation focus and the Cluster Profitability evaluation factor for producer evaluation focus. For consumer evaluation focus, user-centric evaluation factors should consider or constitute QoS attributes that include time, cost, reliability or trust/security (as described in the QoS attributes sub-taxonomy) in order to assess whether the QoS required by consumers is attained.

Market-based RMSs should be evaluated using both system-centric and user-centric evaluation factors to accurately determine its effectiveness in satisfying both system and participant needs.

3) Overhead Analysis: The *overhead analysis* examines potential overheads that are incurred by the market-based RMS. Overheads result in system slowdowns and may create bottlenecks, thus leading to poor efficiency. So, there is a need to evaluate the overheads introduced by the market-based RMS so as to ensure that the overheads are kept to the minimum or within manageable limits.

The *communication* overhead analysis measures the bandwidth overhead incurred due to the communications initiated by the market-based RMS. A high communication overhead incurs higher communication time and can result in unnecessary high network traffic that can slow down data transfers for executions. The *management* overhead analysis calculates the processing overhead sustained in order to derive the resource assignment decisions. Having a high management overhead means that a longer time is required to make resource assignment decisions and thus the market-based RMS is not efficient to support a large number of simultaneous incoming requests. Therefore, market-based RMSs should minimize communication and management overheads in order to be more scalable and efficient.

VI. Survey

Table I shows a summary listing of existing market-based RMSs categorized by different computing platforms, along with their adopted economic models. For our survey, we use the taxonomy to analyze and examine the applicability and suitability of existing market-based RMSs for supporting utility-driven cluster computing. However, we restrict our survey to only some selected market-based RMSs (denoted by * in Table I) that is sufficient to demonstrate how the taxonomy can be applied effectively.

The survey using the various sub-taxonomies are summarized in the following tables: market model (Table II), resource model (Table III), job model (Table IV), resource allocation model (Table V), and performance model (Table VI). The "NA" keyword in the tables denotes that a particular RMS does not address or support the described sub-taxonomy.

TABLE I: Summary of market-based resource management systems

Computing Platform	**Market-based RMS**	**Economic Model**	**Brief Description**
Clusters	Cluster-On-Demand * [20]	tendering/ contract-net	each cluster manager uses a heuristic to measure and balance the future risk of profit lost for accepting a job later against profit gained for accepting the job now.
	Enhanced MOSIX * [19]	commodity market	it uses process migration to minimize the overall execution cost of machines in the cluster.
	Libra * [9]	commodity market	it provides incentives to encourage users to submit job requests with longer deadlines.
	REXEC * [7]	bid-based proportional resource sharing	it allocates resources proportionally to competing jobs based on their users' valuation.
	Utility Data Center [52]	auction	it compares two extreme auction-based resource allocation mechanisms: a globally optimal assignment market mechanism with a sub-optimal simple market mechanism.

TABLE I: Continued.

Computing Platform	Market-based RMS	Economic Model	Brief Description
Agents	D'Agents [17]	bid-based proportional resource sharing	the server assigns resources by computing the clearing price based on the aggregate demand function of all its incoming agents.
	Preist et al. [18]	auction	an agent participates in mutiple auctions selling the same goods in order to secure the lowest bid possible to acquire suitable number of goods for a buyer.
	WALRAS [16]	auction	consumer and producer agents submit their demand and supply curves respectively for a good and the equilibrium price is determined through an iterative auctioning process.
Distributed Databases	Anastasiadi et al. [53]	posted price	it examines the scenario of load balancing economy where servers advertise prices at a bulletin board and transaction requests are routed based on three different routing algorithms that focuses on expected completion time and required network bandwidth.
	Mariposa [6]	tendering/ contract-net	it completes a query within its user-defined budget by contracting portions of the query to various processing sites for execution.
Grids	Bellagio [54]	auction	a centralized auctioneer computes bid values based on number of requested resources and their required durations, before clearing the auctions at fixed time periods by allocating to higher bid values first.
	CATNET [55]	bargaining	each client uses a subjective market price (computing using price quotes consolidated from available servers) to negotiate until a server quotes an acceptable price.
	Faucets * [21]	tendering/ contract-net	users specify QoS contracts for adaptive parallel jobs and Grid resources compete for jobs via bidding.
	G-commerce [56]	commodity market, auction	it compares resource allocation using either commodity market or auction strategy based on four criteria: price stability, market equilibrium, consumer efficiency, and producer efficiency.
	Gridbus [15]	commodity market	it considers the data access and transfer costs for data-oriented applications when allocating resources based on time or cost optimization.
	Gridmarket [57]	auction	it examines resource allocation using double auction where consumers set ceiling prices and sellers set floor prices.
	Grosu and Das [58]	auction	it studies resource allocation using first-price, vickrey and double auctions.
	Maheswaran et al. [59]	auction	it investigates resource allocation based on two "co-bid" approaches that aggregate similar resources: first or no preference approaches.
	Nimrod/G * [8]	commodity market	it allocates resources to task farming applications using either time or cost optimization with deadline and budget constrained algorithms.

TABLE I: Continued.

Computing Platform	Market-based RMS	Economic Model	Brief Description
	OCEAN [60]	bargaining, tendering/ contract-net	it first discovers potential sellers by announcing a buyer's trade proposal and then allows the buyer to determine the best seller by using two possible negotiation mechanisms: yes/no and static bargain.
	Tycoon * [61]	auction	it allocates resources using "auction share" that estimates proportional share with consideration for latency-sensitive and risk-averse applications.
Parallel and Distributed Systems	Agoric Systems [62]	auction	it employs the "escalator" algorithm where users submit bids that escalates over time based on a rate and the server uses vickrey auction at fixed intervals to award resources to the highest bidder who is then charged with the second-highest bid.
	Dynasty [63]	commodity market	it uses a hierarchical-based brokering system where each request is distributed up the hierarchy until the accumulated brokerage cost is limited by the budget of the user.
	Enterprise [64]	tendering/ contract-net	clients broadcast a request for bids with task description and select the best bid which is the shortest estimated completion time given by available servers.
	Ferguson et al. [65]	posted price, auction	it examines how first-price and dutch auctions can support a load balancing economy where each server host its independent auction and users decide which auction to participate based on last clearing prices advertised in bulletin boards.
	Kurose and Simha [66]	bid-based proportional resource sharing	it uses a resource-directed approach where the current allocation of a resource is readjusted proportionally according to the marginal values computed by every agent using that resource to reflect the outstanding quantity of resource needed.
	MarketNet [67]	posted price	it advertises resource request and offer prices on a bulletin board and uses currency flow to restrict resource usage so that potential intrusion attacks into the information systems are controlled and damages caused are kept to the minimum.
	Spawn [5]	auction	it sub-divides each tree-based concurrent program into nodes (sub-programs) which then hold vickrey auction independently to obtain resources.
	Stoica et al. [68]	auction	the job with the highest bid starts execution instantly if the required number of resources are available; else it is scheduled to wait for more resources to be available and has to pay for holding on to currently available resources.
Peer-to-Peer	Stanford Peers * [69]	auction, bartering	it uses data trading to create a replication network of digital archives where a winning remote site offers the lowest bid for free space on the local site in exchange for the amount of free space requested by the local site on the remote site.
World Wide Web	Java Market [70]	commodity market	it uses a cost-benefit framework to host an internet-wide computational market where producers (machines) are paid for executing consumers' jobs (Java programs) as Java applets in their web browsers.

TABLE I: Continued.

Computing Platform	Market-based RMS	Economic Model	Brief Description
	JaWS [28]	auction	it uses double auction to award a lease contract between a client and a host that contains the following information: agreed price, lease duration, compensation, performance statistics vector, and abort ratio.
	POPCORN [27]	auction	each buyer (parallel programs written using POPCORN paradigm) submits a price bid and the winner is determined through one of three implemented auction mechanisms: vickrey, double, and clearinghouse double auctions.
	SuperWeb [26]	commodity market	potential hosts register with client brokers and receive payments for executing Java codes depending on the QoS provided.
	Xenoservers [71]	commodity market	it supports accounted execution of untrusted programs such as Java over the web where resources utilized by the programs are accounted and charged to the users.

TABLE II: Survey using market model taxonomy

Market-based RMS	Economic Model	Participant Focus	Trading Environment	QoS Attributes
Cluster-On-Demand	tendering/ contract-net	producer	competitive	cost
Enhanced MOSIX	commodity market	producer	cooperative	cost
Libra	commodity market	consumer	cooperative	time, cost
REXEC	bid-based proportional resource sharing	consumer	competitive	cost
Faucets	tendering/ contract-net	producer	competitive	time, cost
Nimrod/G	commodity market	consumer	competitive	time, cost
Tycoon	auction	consumer	competitive	time, cost
Stanford Peers	auction, bartering	consumer, producer	cooperative	cost

TABLE III: Survey using resource model taxonomy

Market-based RMS	Management Control	Resource Composition	Execution Service	Execution Support	Accounting Mechanism
Cluster-On-Demand	decentralized	NA	NA	NA	decentralized
Enhanced MOSIX	decentralized	heterogeneous	dedicated	time-shared	decentralized

TABLE III: Continued.

Market-based RMS	Management Control	Resource Composition	Execution Service	Execution Support	Accounting Mechanism
Libra	centralized	heterogeneous	dedicated	time-shared	centralized
REXEC	decentralized	NA	non-dedicated	time-shared	centralized
Faucets	centralized	NA	NA	time-shared	centralized
Nimrod/G	decentralized	heterogeneous	non-dedicated	NA	decentralized
Tycoon	decentralized	heterogeneous	dedicated	time-shared	decentralized
Stanford Peers	decentralized	NA	dedicated	NA	NA

TABLE IV: Survey using job model taxonomy

Market-based RMS	Job Execution	Job Dependency	Job Composition	QoS Specification	QoS Update
Cluster-On-Demand	sequential	NA	single-task	rate-based	static
Enhanced MOSIX	parallel	NA	NA	NA	NA
Libra	sequential	NA	single-task	constraint-based	static
REXEC	parallel, sequential	NA	single-task	constraint-based	static
Faucets	parallel	NA	NA	constraint-based	static
Nimrod/G	sequential	NA	mutiple-task	optimization-based	static
Tycoon	NA	NA	NA	constraint-based	static
Stanford Peers	NA	NA	NA	NA	NA

TABLE V: Survey using resource allocation model taxonomy

Market-based RMS	Resource Allocation Domain	Resource Allocation Update	QoS Support
Cluster-On-Demand	external	non-adaptive	soft
Enhanced MOSIX	internal	adaptive	NA
Libra	internal	adaptive	hard
REXEC	internal	adaptive	hard
Faucets	internal	adaptive	soft
Nimrod/G	external	adaptive	soft
Tycoon	internal	adaptive	soft

TABLE V: Continued.

Market-based RMS	Resource Allocation Domain	Resource Allocation Update	QoS Support
Stanford Peers	external	non-adaptive	NA

TABLE VI: Survey using performance model taxonomy

Market-based RMS	Evaluation Focus	Evaluation Factors	Overhead Analysis
Cluster-On-Demand	producer	user-centric (cost)	NA
Enhanced MOSIX	consumer	user-centric (time)	NA
Libra	consumer, producer	system-centric, user-centric (time, cost)	NA
REXEC	consumer	user-centric (cost)	NA
Faucets	NA	NA	NA
Nimrod/G	NA	NA	NA
Tycoon	consumer	user-centric (time)	communication, management
Stanford Peers	consumer, producer	user-centric (reliability)	NA

A. Cluster-On-Demand

Cluster-On-Demand (COD) [72] allows the cluster manager to dynamically create independent partitions called virtual clusters (vclusters) with specific software environments for each different user groups within a cluster system. This in turn facilitates external policy managers and resource brokers in the Grid to control their assigned vcluster of resources. A later work [20] examines the importance of opportunity cost in a service market where earnings for a job depreciates linearly over increasing time delay. A falling earning can become zero and instead become a penalty for not fulfilling the contract of task execution. Thus, each local cluster manager needs to determine the best job mix to balance the gains and losses for selecting a task instead of other tasks.

The task assignment among various cluster managers adopts the tendering/contract-net economic model. A user initiates an announcement bid that reflects its valuation for the task to all the cluster managers. Each cluster manager then considers the opportunity cost (gain or loss) for accepting the task and proposes a contract with an expected completion time and price. The user then selects and accepts a contract from the cluster manager which responded.

A competitive trading environment with producer participant focus is supported since each cluster manager aims to maximize its own earnings by accessing the risk and reward for bidding and scheduling a task. Earnings are paid by users to cluster managers as costs for adhering to the conditions of the contract. All cluster managers maintain information about its committed workload in order to evaluate whether to accept or reject a new task, hence exercising decentralized management control and accounting mechanism.

Tasks to be executed are assumed to single and sequential. For each task, the user provides a value function containing a constant depreciation rate to signify the importance of the task and thus the required level of service. The value function remains static after the contract has been accepted by the user. Tasks are scheduled externally to cluster managers in different administrative domains. Non-adaptive resource allocation update is supported as the cluster manager which is awarded the contract has to ensure the completion of the task. However, the completion time of a task varies as the cluster manager may delay less costly committed tasks for more costly new tasks to maximize its profit, thus providing soft QoS support.

Performance evaluation focuses on producer by using a user-centric cost evaluation factor to determine the average yield or earning each cluster manager achieves. Simulation results shows that considering and balancing the potential gain of accepting a task instantly with the risk of future loss provides better returns for competing cluster managers.

B. Enhanced MOSIX

Enhanced MOSIX [19] is a modified version of MOSIX [73] cluster operating system that employs an opportunity cost approach for load balancing to minimize the overall execution cost of the cluster. The opportunity cost approach computes a single marginal cost of assigning a process to a cluster node based on the processor and memory usages of the process, thus representing a commodity market economic model. The cluster node with the minimal marginal cost is then assigned the process. This implies a cooperative trading environment with producer participant focus whereby the cost utility is measured in terms of usage level of resources.

In Enhanced MOSIX, decentralized resource control is established where each cluster node makes its independent resource assignment decisions. Heterogeneous resource composition is supported by translating usages of different resources into a single cost measure. Dedicated execution service is inferred as MOSIX is a cluster operating system and is aware of all active executing processes.

Enhanced MOSIX supports a time-sharing parallel execution environment where a user can execute a parallel application by first starting multiple processes on one cluster node. Each cluster node maintains accounting information about processes on its node and exchange information with other nodes periodically to determine which processes can be migrated based on the opportunity cost approach. Process migration is utilized internally within the cluster to assign or reassign processes to less loaded nodes, hence supporting adaptive resource allocation update.

Enhanced MOSIX does not address how QoS can be supported for users. For performance evaluation, it measures the slowdown of user processes, hence using a user-centric time evaluation factor. Simulation results show that using the opportunity cost approach returns a lower average slowdown of processes, thus benefiting the consumers.

C. Libra

Libra [9] is designed to be a pluggable market-based scheduler that can be integrated into existing cluster RMS architectures to support allocation of resources based on users' QoS requirements. Libra adopts the commodity market economic model that charges users using a pricing function. A later work [51] proposes an enhanced pricing function that supports four essential requirements for pricing of utility-driven cluster resources: flexible, fair, dynamic, and adaptive.

The pricing function is flexible to allow easy configuration of the cluster owner to determine the level of sharing. It is also fair as resources are priced based on actual usage; jobs that use more resources are charged more. The price of resources is dynamic and is not based on a static rate. In addition, the price of resources adapts to the changing supply and demand of resources. For instance, high cluster workload results in increased pricing to discourage users from submitting infinitely and thus not overloading the cluster. This is crucial in providing QoS support since an overloaded cluster will not be able to fulfill QoS requirements. In addition, incentive is given to promote users to submit jobs with longer deadlines; a job with longer deadline is charged less compared to a job with shorter deadline.

The main objective of Libra is to maximize the number of jobs whose QoS requirements can be met, thus enabling a consumer participant focus. The enhanced pricing function [51] also improves utility for the producer (cluster owner) as only jobs with higher budgets are accepted with increasing cluster workload. Libra also considers both time and cost QoS attributes by allocating resources based on the deadline and budget QoS parameters for each job. A cooperative trading environment is implied as users are encouraged to provide a more relaxed deadline through incentives so that more jobs can be accommodated.

Libra communicates with the centralized resource manager in the underlying cluster RMS that collects information about resources in the cluster. For heterogeneous resource composition, measures such as estimated execution time are translated to their equivalent on different worker nodes. The cluster RMS also provides dedicated execution service as it is the only gateway for users to submit jobs into the cluster and thus is aware of all jobs active in the cluster. The cluster RMS needs to support time-shared execution given that Libra allocates resources to multiple executing jobs based on their required deadline. This ensures that a more urgent job with shorter remaining time to its deadline is allocated a larger processor time partition on a worker node as compared to a less urgent job. Libra uses a centralized accounting mechanism to monitor resource usage of active jobs so as to periodically reallocate the time partitions for each active job to ensure all jobs still complete within their required deadline.

Libra currently assumes that submitted jobs are sequential and single-task. Users can express two QoS constraints: deadline which the job needs to be completed and budget which the user is willing to pay. The QoS constraints cannot be updated after the job has been accepted for execution. Libra only schedules jobs to internal worker nodes within the cluster system. Each worker node has a job control component that reassigns processor time partitions periodically based on the actual execution and required deadline of each active job, thus enforcing hard QoS support.

Libra uses average waiting time and average response time as system-centric evaluation factors to evaluate overall system performance. In addition, Libra defines two user-centric evaluation factors [51]: Job QoS Satisfaction and Cluster Profitability to measure the level of utility achieved for the consumers (users) and producer (cluster owner) respectively. The Job QoS Satisfaction determines the percentage of jobs whose deadline and budget QoS is satisfied and thus examines the time and cost utility of the consumers. On the other hand, the Cluster Profitability calculates the proportion of profit obtained by the cluster owner and thus studies the cost utility of the producer. Simulation results show that Libra performs better than traditional First-Come-First-Served scheduling approach for both system-centric and user-centric evaluation factors.

D. REXEC

REXEC [7] implements bid-based proportional resource sharing where users compete for shared resources in a cluster. It has a consumer participant focus since resources are allocated proportionally based on costs that competing users are willing to pay for a resource. Costs are defined as rates, such as credits per minute to reflect the maximum amount that a user wants to pay for using the resource.

Decentralized management control is achieved by having multiple daemons to separately discover and determine the best node to execute a job and then allowing each REXEC client to directly manage the execution of its jobs on the selected cluster nodes. The cluster nodes can be non-dedicated as nodes are discovered dynamically during scheduling and supports time-shared execution support so that multiple jobs share resources at the same time. A centralized accounting service maintains credit usage for each user in the cluster. REXEC does not consider the resource composition since it determines the proportion of resource assignment for a job purely on its user's valuation.

REXEC supports the execution of both sequential and parallel programs. Users specify constraint-based cost limits that they are willing to spend and remains static after job submission. The discovery and selection of nodes internal in the cluster system is designed to be independent so that users have the flexibility to determine the node selection policy through their own REXEC client. Existing resource assignments are recomputed whenever a new job starts or finishes on a node, thus enabling adaptive resource allocation update. REXEC only considers a single QoS where the cost of job execution is limited to the users' specified rate. For a parallel program, the total credit required by all its processes is enforced not to exceed the cost specified by the user.

A later work [74] uses a user-centric evaluation factor: aggregate utility that adds up all the users' costs for completing jobs on the cluster. The cost charged to the user depends on the completion time of his job and decreases linearly over time until it reaches zero. Therefore, this presents a consumer evaluation focus where cost is the evaluation factor.

E. Faucets

Faucets [21] aims to provide efficient resource allocation on the computational Grid for parallel jobs by improving its usability and utilization. For better usability, users will not need to manually discover the best resources to execute their jobs or monitor the progress of executing jobs. To improve utilization, the parallel jobs are made adaptive using Charm++ [75] or adaptive MPI [76] frameworks so that they can be executed on changing number of allocated processors during runtime on demand [77]. This allows more jobs to be executed at any one time and no processors are left unused.

Market economy is implemented to promote utilization of the computational Grid where each individual Grid resource maximizes its profit through maximum resource utilization. For each parallel job submitted, the user has to specify its QoS contract that includes requirements such as the software environment, number of processors (can be a single number, a set of numbers or range of numbers), expected completion time (and how this changes with number of processors), and the payoff that the user will pay to the Grid resource (and how this changes with actual job completion time). With this QoS contract, a parallel job completed by Faucets will have three possible economic outcomes: payoff at soft deadline, a decreased payoff at hard deadline (after soft deadline) and penalty after hard deadline.

Faucets uses the tendering/contract-net market economic model. First, it determines the list of Grid resources that are able to satisfy the job's execution requirements. Then, requests are sent out to each of these Grid resources to inform them about this new job. Grid resources can choose to decline or reply with a bid. The user then chooses the Grid resource when all the bids are collected.

Faucets has a producer participant focus and competitive trading environment as each Grid resource aims to maximize its own profit and resource utilization and thus compete with other resources. Faucets considers the time QoS attribute since each Grid resource that receives a new job request first checks that it can satisfy the job's QoS contract before replying with a bid. The cost QoS attribute is decided by the user who then chooses the resource to execute based on the bids of the Grid resources.

Faucets currently uses a centralized management control where the Faucets Central Server (FS) maintains the list of resources and applications that user can execute. However, the ultimate aim of Faucets is to have a distributed management control to improve scalability. Time-shared execution support is employed in Faucets where adaptive jobs executes simultaneously but on different proportion of allocated processors. A centralized accounting mechanism at the FS keeps track of participating

Grid resources so that owners of these Grid resources can earn credits to execute jobs on other Grid resources. Faucets is primarily designed to support parallel job execution only where the constraint-based QoS contract of a parallel job is given at job submission and remains static throughout the execution.

In Faucets, the resource allocation domain operates in an internal manner where each Grid resource is only aware of jobs submitted via the FS and not other remote Grid resources. To maximize system utilization at each Grid resource, Faucets allocates proportional number of processors to jobs based on their QoS priorities since jobs are adaptive to changing number of processors. A new job with higher priority is allocated a larger proportion of processors, thus resulting in existing jobs entitled to shrinking proportion of processors. This results in soft QoS support. Faucets does not describe how utility-driven performance can be evaluated.

F. Nimrod/G

Nimrod/G [8] is the grid-enabled version of Nimrod [78] that allows user to create and execute parameter sweep applications on the Grid. Its design is based on a commodity market economic model where each Nimrod/G broker associated with a user obtains service prices from Grid traders at each different Grid resource location. Nimrod/G supports a consumer participant focus that considers deadline (time) and budget (cost) QoS constraints specified by the user for running his application. Prices of resources thus vary between different executing applications depending on the time and selection of Grid resources that suits the QoS constraints. This means that users have to compete with one another in order to maximize their own personal benefits, thus establishing a competitive trading environment.

Each Nimrod/G broker acts on behalf of its user to discover the best resources for his application and does not communicate with other brokers, thus implementing a decentralized management control. It also has its own decentralized accounting mechanism to ensure that the multiple tasks within the parameter sweep application will not violate the overall constraints. In addition, the Nimrod/G broker is able to operate in a highly dynamic Grid environment where resources are heterogeneous and non-dedicated since they are managed by different owners, each having their own operating policies. The broker does not need to know the execution support of each Grid resource as each resource will feedback to the broker their estimated completion time for a task.

A parameter sweep application generates multiple independent tasks with different parameter values that can execute sequentially on a processor. For each parameter sweep application, the Nimrod/G broker creates a plan to assign tasks to resources that either optimizes time or cost within deadline and budget constraints or only satisfies the constraints without any optimization [79]. The QoS constraints for a parameter sweep application can only be specified before the creation of the plan and remains static when the resource broker discovers and schedules suitable resources.

The Nimrod/G broker discovers external Grid resources across multiple administrative domains. Resources are discovered and assigned progressively for the multiple tasks within an application depending on current resource availability that is beyond the control of the broker. Therefore, Nimrod/G is only able to provide soft QoS support as it tries its best to fulfill the QoS constraints. It supports some level of adaptive resource allocation update as it attempts to discover resources for remaining tasks yet to be scheduled based on the remaining budget from scheduled tasks so that the overall budget is not exceeded. It will also attempt to reschedule tasks to other resources if existing scheduled tasks fails to start execution. However, Nimrod/G will stop assigning remaining tasks once the deadline or budget QoS constraint is violated, thus wasting budget and time spent on already completed tasks. Nimrod/G does not describe how utility-driven performance can be evaluated.

G. Tycoon

Tycoon [61] examines resource allocation in Grid environments where users are self-interested with unpredictable demands and service hosts are unreliable with changing availability. It implements a two-tier resource allocation architecture that differentiates between user strategy and allocation mechanism. The user strategy captures high-level preferences that are application-dependent and vary across users, while the allocation mechanism provides means to solicit true user valuations for more efficient execution. The separation of user strategy and allocation mechanism therefore allows both requirements not to be limited and dependent of one another.

Each service host utilizes an auction share scheduler that holds first-price auctions to determine resource allocation. The request with the highest bid is then allocated the processor time slice. The bid is computed as the pricing rate that the user will pay for the required processor time, hence both time and cost QoS attributes are considered. Consumer participant focus is supported as users can indicate whether the service requests are latency-sensitive or throughput-driven. Based on these preferences, consumers have to compete with one another for Grid sites that can satisfy their service requests.

A host self-manages its local selection of applications, thus maintaining decentralized resource management. Hosts are heterogeneous since they are installed in various administrative domains and owned by different owners. The hosts provide dedicated execution service by accepting service requests submitted via the Tycoon interface. Applications are assigned processor time slices so that multiple requests can be concurrently executed. Each host also keeps accounting information

of its local applications to calculate the usage-based service cost to be paid by the user and determine prices of future resource reservation for risk-averse applications.

Tycoon is assumed to handle general service applications that include web and database services. Service execution requests are specified in terms of constraints such as the amount of cost he plans to spend and the deadline for completion. These constraints do not change after initial specification. Each auction share scheduler performs resource assignment internally within the service host. It also enables adaptive resource allocation update as new service requests will modify and reduce the current resource entitlements of existing executing requests. This results in soft QoS support that can have a negative impact for risk-averse and latency-sensitive applications. To minimize this, Tycoon allows users to reserve resources in advance to ensure sufficient entitlements.

The performance evaluation concentrates on consumer using a user-centric time evaluation factor. A metric called scheduling error assesses whether users get their specified amount of resources and also justifies the overall fairness for all users. The mean latency is also measured for latency-sensitive applications to examine whether their requests are fulfilled. Simulation results show that the Tycoon is able to achieve high fairness and low latency compared to simple proportional-share strategy. Tycoon has addressed how communication and management overheads are designed to be minimal. For instance, auctions held internally within each service host reduce communication across hosts.

H. Stanford Peers

Stanford Peers [69] employs a peer-to-peer data trading framework to create a digital archiving system. It utilizes a bid trading auction mechanism where a local site that wants to replicate its collection holds an auction to solicit bids from remote sites by first announcing its request for storage space. Each interested remote site then returns a bid that reflects the amount of disk storage space to request from the local site in return for providing the requested storage space. The local site then selects the remote site with the lowest bid for maximum benefit.

An overall cooperative trading environment with both producer and consumer participant focus is supported as a bartering system is built whereby sites exchange free storage spaces to benefit both themselves and others. Each site minimizes the cost of trading which is the amount of disk storage space it has to provide to the remote site for the requested data exchange. Stanford Peers implements decentralized management control as each site makes its own decision to select the most suitable remote sites to replicate its data collection.

Sites are dedicated to protect the replicated data in order to ensure that data are preserved and accessible. Each site is external of one another and can belong to different owners. Once a remote site is selected, the specified amount of storage space remains fixed, hence implying non-adaptive resource allocation update. The job model taxonomy does not apply to Stanford Peers because the allocation of resources is expressed in terms of data exchange and not jobs.

Stanford Peers evaluates performance based on reliability against failures since the focus of a archiving system is to preserve data as long as possible. Reliability is measured using the mean time to failure (MTTF) for each local site that is both a producer and consumer. Simulation results show that sites that uses bid trading achieves higher reliability than sites that trade equal amounts of space without bidding.

VII. Discussion and Conclusion

There are a few market-based RMSs implemented for cluster computing such as Cluster-On-Demand [20], Enhanced MOSIX [19], Libra [9], REXEC [7] and Utility Data Center [52]. Libra and REXEC provide a consumer participant focus that is crucial for satisfying QoS-based requests in service-oriented cluster and Grid computing. However, none of these market-based RMSs supports other important QoS attributes such as reliability and trust/security that should be realized in a utility-driven service market where consumers pay for usage and expect good quality service.

These market-based RMSs for cluster systems only support fairly simple job models with sequential job execution, single-task job composition and static QoS update. They do not allow more advanced job models with data/sequence dependencies such as in workflow-based applications, multiple-task composition in parameter sweep applications and parallel execution in message-passing applications. Users may also need to modify their initial QoS specifications after job submission and thus require the support of dynamic QoS update. In addition, the scope of resource allocation is often restricted to internally within the cluster systems. They can be extended to discover and utilize external resources in other cluster systems or Grids so that a larger pool of resources is available for usage. For performance evaluation, both system-centric and user-centric evaluation factors should be defined to measure the effectiveness of market-based cluster RMSs in achieving overall system performance and actual benefits for both consumers and producers. Metrics should also be defined to measure communication and management overheads incurred by the market-based RMSs.

Market-based RMSs proposed for other computing platforms encompass strengths that can be leveraged for the context of cluster computing. For instance, the tendering/contract-net economic model in Faucets [21] may be applied in a cluster system with decentralized management control where the consumer determines the resource selection by choosing the best node based on bids from competing cluster nodes. Optimization-based QoS specification in Nimrod/G [8] and the “auction share”

scheduling algorithm in Tycoon [61] can improve utility for consumers, in particular those with latency-sensitive applications. Bartering concepts in Stanford Peers [69] can augment the level of sharing across internal and external resource allocation domains.

In this paper, we have described how market-based RMSs can achieve the requirements of utility-driven cluster computing. We have outlined an abstract model capturing essential functionalities of a market-based cluster RMS and developed a taxonomy classifying market-based RMSs to support utility in cluster systems. We have also applied the taxonomy to survey some recent market-based RMSs proposed for both cluster computing and other computing platforms to identify possible future enhancements.

ACKNOWLEDGMENT

We would like to acknowledge all developers of the market-based resource management systems described in the paper. This work is partially supported by the Australian Research Council (ARC) Discovery Project and StorageTek Fellowship for Grid Computing.

REFERENCES

[1] W. Gropp, E. Lusk, and T. Sterling, Eds., *Beowulf Cluster Computing with Linux*, 2nd ed. Cambridge, MA: MIT Press, 2003.
[2] G. F. Pfister, *In Search of Clusters*, 2nd ed. Upper Saddle River, NJ: Prentice Hall PTR, 1998.
[3] R. Buyya, Ed., *High Performance Cluster Computing: Architectures and Systems*. Upper Saddle River, NJ: Prentice Hall PTR, 1999.
[4] R. Buyya, T. Cortes, and H. Jin, "Single System Image," *The International Journal of High Performance Computing Applications*, vol. 15, no. 2, pp. 124–135, Summer 2001.
[5] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta, "Spawn: A Distributed Computational Economy," *IEEE Trans. Software Eng.*, vol. 18, no. 2, pp. 103–117, Feb. 1992.
[6] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin, "An economic paradigm for query processing and data migration in Mariposa," in *Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems (PDIS '94)*. Austin, TX: IEEE Computer Society Press: Los Alamitos, CA, Sept. 1994, pp. 58–68.
[7] B. N. Chun and D. E. Culler, "Market-based Proportional Resource Sharing for Clusters," Computer Science Division, University of California at Berkeley, Technical Report CSD-1092, Jan. 2000.
[8] D. Abramson, R. Buyya, and J. Giddy, "A computational economy for grid computing and its implementation in the Nimrod-G resource broker," *Future Generation Computer Systems*, vol. 18, no. 8, pp. 1061–1074, Oct. 2002.
[9] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya, "Libra: a computational economy-based job scheduling system for clusters," *Software: Practice and Experience*, vol. 34, no. 6, pp. 573–590, May 2004.
[10] I. Foster and C. Kesselman, Eds., *The Grid 2: Blueprint for a New Computing Infrastructure*. San Francisco, CA: Morgan Kaufmann, 2003.
[11] R. Buyya, D. Abramson, and J. Giddy, "A Case for Economy Grid Architecture for Service Oriented Grid Computing," in *Proceedings of the 10th International Heterogeneous Computing Workshop (HCW 2001)*. San Francisco, CA: IEEE Computer Society Press: Los Alamitos, CA, Apr. 2001.
[12] ——, "Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid," in *Proceedings of the 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*. Beijing, China: IEEE Computer Society Press: Los Alamitos, CA, May 2000.
[13] S. Venugopal, R. Buyya, and L. Winton, "A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids," in *Proceedings of the 2nd International Workshop on Middleware for Grid Computing (MGC 2004)*. Toronto, Canada: ACM Press: New York, NY, Oct. 2004, pp. 75–80.
[14] J. Yu and R. Buyya, "A Novel Architecture for Realizing Grid Workflow Using Tuple Spaces," in *Proceedings of the 5th International Workshop on Grid Computing (GRID 2004)*. Pittsburgh, PA: IEEE Computer Society Press: Los Alamitos, CA, Nov. 2004, pp. 119–128.
[15] R. Buyya, D. Abramson, and S. Venugopal, "The Grid Economy," *Proc. IEEE*, vol. ?, no. ?, p. (to appear), Dec. 2004.
[16] M. P. Wellman, "A Market-Oriented Programming Environment and its Application to Distributed Multicommodity Flow Problems," *Journal of Artificial Intelligence Research*, vol. 1, pp. 1–23, 1993.
[17] J. Bredin, D. Kotz, and D. Rus, "Utility Driven Mobile-Agent Scheduling," Department of Computer Science, Dartmouth College, Technical Report PCS-TR98-331, Oct. 1998.
[18] C. Preist, A. Byde, and C. Bartolini, "Economic dynamics of agents in multiple auctions," in *Proceedings of the 5th International Conference on Autonomous Agents (AGENTS 2001)*. Montreal, Canada: ACM Press: New York, NY, May–June 2001, pp. 545–551.
[19] Y. Amir, B. Awerbuch, A. Barak, R. S. Borgstrom, and A. Keren, "An Opportunity Cost Approach for Job Assignment in a Scalable Computing Cluster," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 7, pp. 760–768, July 2000.
[20] D. E. Irwin, L. E. Grit, and J. S. Chase, "Balancing Risk and Reward in a Market-based Task Service," in *Proceedings of the 13th International Symposium on High Performance Distributed Computing (HPDC13)*. Honolulu, HI: IEEE Computer Society Press: Los Alamitos, CA, June 2004, pp. 160–169.
[21] L. V. Kalé, S. Kumar, M. Potnuru, J. DeSouza, and S. Bandhakavi, "Faucets: Efficient Resource Allocation on the Computational Grid," in *Proceedings of the International Conference on Parallel Processing (ICPP 2004)*. Montreal, Canada: IEEE Computer Society Press: Los Alamitos, CA, Aug. 2004, pp. 396–405.
[22] R. Cocchi, D. Estrin, S. Shenker, and L. Zhang, "A Study of Priority Pricing in Multiple Service Class Networks," *ACM SIGCOMM Computer Communication Review*, vol. 21, no. 4, pp. 123–130, Sept. 1991.
[23] D. D. Clark, "A Model for Cost Allocation and Pricing in the Internet," *Journal of Electronic Publishing*, vol. 2, no. 1, May 1996. [Online]. Available: http://www.press.umich.edu/jep/works/ClarkModel.html
[24] S. Shenker, D. Clark, D. Estrin, and S. Herzog, "Pricing in Computer Networks: Reshaping the Research Agenda," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 2, pp. 19–43, Apr. 1996.
[25] R. Edell and P. Varaiya, "Providing Internet Access: What We Learn From INDEX," *IEEE Network*, vol. 13, no. 5, pp. 18–25, Sept.–Oct. 1999.
[26] A. D. Alexandrov, M. Ibel, K. E. Schauser, and C. J. Scheiman, "SuperWeb: research issues in Java-based global computing," *Concurrency: Practice and Experience*, vol. 9, no. 6, pp. 535–553, June 1997.
[27] O. Regev and N. Nisan, "The POPCORN Market - an Online Market for Computational Resources," in *Proceedings of the 1st International Conference on Information and Computation Economies (ICE '98)*. Charleston, SC: ACM Press: New York, NY, Oct. 1998, pp. 148–157.

[28] S. Lalis and A. Karipidis, "JaWS: An Open Market-Based Framework for Distributed Computing over the Internet," in *Proceedings of the 1st International Workshop on Grid Computing (GRID 2000)*, ser. Lecture Notes in Computer Science (LNCS), vol. 1971/2000. Bangalore, India: Springer Verlag: Heidelberg, Germany, Dec. 2000, pp. 36–46.

[29] T. L. Casavant and J. G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," *IEEE Trans. Software Eng.*, vol. 14, no. 2, pp. 141–154, Feb. 1988.

[30] H. G. Rotithor, "Taxonomy of dynamic task scheduling schemes in distributed computing systems," *IEE Proceedings of Computers and Digital Techniques*, vol. 141, no. 1, pp. 1–10, Jan. 1994.

[31] I. Ekmecić, I. Tartalja, and V. Milutinović, "EM^3: A taxonomy of heterogeneous computing systems," *IEEE Computer*, vol. 28, no. 12, pp. 68–70, Dec. 1995.

[32] ——, "A survey of heterogeneous computing: concepts and systems," *Proc. IEEE*, vol. 84, no. 8, pp. 1127–1144, Aug. 1996.

[33] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing systems," in *Proceedings of the 17th Symposium on Reliable Distributed Systems*. West Lafayette, IN: IEEE Computer Society Press: Los Alamitos, CA, Oct. 1998, pp. 330–335.

[34] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," *Software: Practice and Experience*, vol. 32, no. 2, pp. 135–164, Feb. 2002.

[35] *Condor Version 6.7.1 Manual*, University of Wisconsin-Madison, 2004. [Online]. Available: http://www.cs.wisc.edu/condor/manual/v6.7

[36] *LoadLeveler for AIX 5L Version 3.2 Using and Administering*, SA22-7881-01, IBM, Oct. 2003. [Online]. Available: http://www-1.ibm.com/servers/eserver/pseries/library/sp_books/loadleveler.html

[37] *LSF Version 4.1 Administrator's Guide*, Platform Computing, 2001. [Online]. Available: http://www.platform.com/services/support

[38] *OpenPBS Release 2.3 Administrator Guide*, Altair Grid Technologies, Aug. 2000. [Online]. Available: http://www.openpbs.org/docs.html

[39] *Sun ONE Grid Engine, Administration and User's Guide*, Sun Microsystems, Oct. 2002. [Online]. Available: http://gridengine.sunsource.net/project/gridengine/documentation.html

[40] (2004, Nov.) The TeraGrid Project. [Online]. Available: http://www.teragrid.org

[41] (2004, Nov.) The LHC Computing Grid Project. [Online]. Available: http://lcg.web.cern.ch/LCG

[42] (2004, Nov.) The NAREGI Project. [Online]. Available: http://www.naregi.org

[43] (2004, Nov.) The APAC Grid Project. [Online]. Available: http://www.apac.edu.au

[44] (2004, Nov.) IBM Grid Computing. [Online]. Available: http://www.ibm.com/grid

[45] (2004, Nov.) HP Grid Computing. [Online]. Available: http://www.hp.com/techservers/grid

[46] (2004, Nov.) Sun Microsystems Utility Computing. [Online]. Available: http://www.sun.com/service/utility

[47] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic models for resource management and scheduling in Grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13–15, pp. 1507–1542, Nov.–Dec. 2002.

[48] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, "Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework," in *High Performance Computing: Paradigm and Infrastructure*, L. T. Yang and M. Guo, Eds. ?: John Wiley and Sons, ?, ch. ?, pp. –.

[49] A. Barmouta and R. Buyya, "GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration," in *Proceedings of the 3rd Workshop on Internet Computing and E-Commerce (ICEC 2003), International Parallel and Distributed Processing Symposium (IPDPS 2003)*. Nice, France: IEEE Computer Society Press: Los Alamitos, CA, Apr. 2003.

[50] S. M. Jackson, *Allocation Management with QBank*, Pacific Northwest National Laboratory, 2004. [Online]. Available: http://www.emsl.pnl.gov/docs/mscf/qbank

[51] C. S. Yeo and R. Buyya, "Pricing for Utility-driven Resource Management and Allocation in Clusters," in *Proceedings of the 12th International Conference on Advanced Computing and Communication (ADCOM 2004)*. Ahmedabad, India: ?:?, Dec. 2004, p. (to appear).

[52] A. Byde, M. Sallé, and C. Bartolini, "Market-Based Resource Allocation for Utility Data Centers," HP Lab, Bristol, Technical Report HPL-2003-188, Sept. 2003.

[53] A. Anastasiadi, S. Kapidakis, C. Nikolaou, and J. Sairamesh, "A Computational Economy for Dynamic Load Balancing and Data Replication," in *Proceedings of the 1st International Conference on Information and Computation Economies (ICE '98)*. Charleston, SC: ACM Press: New York, NY, Oct. 1998, pp. 166–180.

[54] A. AuYoung, B. N. Chun, A. C. Snoeren, and A. Vahdat, "Resource Allocation in Federated Distributed Computing Infrastructures," in *Proceedings of the 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS 2004)*, Boston, MA, Oct. 2004.

[55] T. Eymann, M. Reinicke, O. Ardaiz, P. Artigas, F. Freitag, and L. Navarro, "Decentralized Resource Allocation in Application Layer Networks," in *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid (CCGrid 2003)*. Tokyo, Japan: IEEE Computer Society Press: Los Alamitos, CA, May 2003, pp. 645–650.

[56] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan, "Analyzing Market-Based Resource Allocation Strategies for the Computational Grid," *International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 258–281, Fall 2001.

[57] M. Chen, G. Yang, and X. Liu, "Gridmarket: A Practical, Efficient Market Balancing Resource for Grid and P2P Computing," in *Proceedings of the 2nd International Workshop on Grid and Cooperative Computing (GCC 2003)*, ser. Lecture Notes in Computer Science (LNCS), vol. 3033/2004. Shanghai, China: Springer Verlag: Heidelberg, Germany, Dec. 2003, pp. 612–619.

[58] D. Grosu and A. Das, "Auction-Based Resource Allocation Protocols in Grids," in *Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS 2004)*. Cambridge, MA: ACTA Press: Calgary, Canada, Nov. 2004, pp. 20–27.

[59] C. Chen, M. Maheswaran, and M. Toulouse, "Supporting Co-allocation in an Auctioning-based Resource Allocator for Grid Systems," in *Proceedings of the 11th International Heterogeneous Computing Workshop (HCW 2002)*. Fort Lauderdale, FL: IEEE Computer Society Press: Los Alamitos, CA, Apr. 2002.

[60] P. Padala, C. Harrison, N. Pelfort, E. Jansen, M. P. Frank, and C. Chokkareddy, "OCEAN: The Open Computation Exchange and Arbitration Network, A Market Approach to Meta Computing," in *Proceedings of the 2nd International Symposium on Parallel and Distributed Computing (ISPDC 2003)*. Ljubljana, Slovenia: IEEE Computer Society Press: Los Alamitos, CA, Oct. 2003, pp. 185–192.

[61] K. Lai, B. A. Huberman, and L. Fine, "Tycoon: A Distributed Market-based Resource Allocation System," HP Lab, Palo Alto, Technical Report cs.DC/0404013, Apr. 2004.

[62] M. S. Miller and K. E. Drexler, "Incentive Engineering for Computational Resource Management," in *The Ecology of Computation*, B. A. Huberman, Ed. New York, NY: Elsevier Science Publisher, 1988, ch. 10, pp. 231–266.

[63] M. Backschat, A. Pfaffinger, and C. Zenger, "Economic-Based Dynamic Load Distribution in Large Workstation Networks," in *Proceedings of the 2nd International Euro-Par Conference (Euro-Par 1996)*, ser. Lecture Notes in Computer Science (LNCS), vol. 1124/1996. Lyon, France: Springer Verlag: Heidelberg, Germany, Aug. 1996, pp. 631–634.

[64] T. W. Malone, R. E. Fikes, K. R. Grant, and M. T. Howard, "Enterprise: A Market-like Task Scheduler for Distributed Computing Environments," in *The Ecology of Computation*, B. A. Huberman, Ed. New York, NY: Elsevier Science Publisher, 1988, pp. 177–205.

[65] D. F. Ferguson, Y. Yemini, and C. Nikolaou, "Microeconomic Algorithms for Load Balancing in Distributed Computer Systems," in *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS '88)*. San Jose, CA: IEEE Computer Society Press: Los Alamitos, CA, June 1988, pp. 491–499.

[66] J. F. Kurose and R. Simha, "A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems," *IEEE Trans. Comput.*, vol. 38, no. 5, pp. 705–717, May 1989.

[67] Y. Yemini, A. Dailianas, D. Florissi, and G. Huberman, "MarketNet: protecting access to information systems through financial market controls," *Decision Support Systems*, vol. 28, no. 1–2, pp. 205–216, Mar. 2000.

[68] I. Stoica, H. Abdel-Wahab, and A. Pothen, "A Microeconomic Scheduler for Parallel Computers," in *Proceedings of the 1st Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '95)*, ser. Lecture Notes in Computer Science (LNCS), vol. 949/1995. Santa Barbara, CA: Springer Verlag: Heidelberg, Germany, Apr. 1995, pp. 200–218.

[69] B. F. Cooper and H. Garcia-Molina, "Bidding for storage space in a peer-to-peer data preservation system," in *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS 2002)*. Vienna, Austria: IEEE Computer Society Press: Los Alamitos, CA, July 2002, pp. 372–381.

[70] Y. Amir, B. Awerbuch, and R. S. Borgstrom, "A Cost-Benefit Framework for Online Management of a Metacomputing System," in *Proceedings of the 1st International Conference on Information and Computation Economies (ICE '98)*. Charleston, SC: ACM Press: New York, NY, Oct. 1998, pp. 140–147.

[71] D. Reed, I. Pratt, P. Menage, S. Early, and N. Stratford, "Xenoservers: Accountable Execution of Untrusted Programs," in *Proceedings of the 7th Workshop on Hot Topics in Operating Systems (HotOS-VII)*. Rio Rico, AZ: IEEE Computer Society Press: Los Alamitos, CA, Mar. 1999, pp. 136–141.

[72] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. E. Sprenkle, "Dynamic Virtual Clusters in a Grid Site Manager," in *Proceedings of the 12th International Symposium on High Performance Distributed Computing (HPDC12)*. Seattle, WA: IEEE Computer Society Press: Los Alamitos, CA, June 2003, pp. 90–100.

[73] A. Barak and O. La'adan, "The MOSIX multicomputer operating system for high performance cluster computing," *Future Generation Computer Systems*, vol. 13, no. 4–5, pp. 361–372, Mar. 1998.

[74] B. N. Chun and D. E. Culler, "User-centric Performance Analysis of Market-based Cluster Batch Schedulers," in *Proceedings of the 2nd International Symposium on Cluster Computing and the Grid (CCGrid 2002)*. Berlin, Germany: IEEE Computer Society Press: Los Alamitos, CA, May 2002, pp. 22–30.

[75] L. V. Kalé and S. Krishnan, "Charm++: Parallel Programming with Message-Driven Objects," in *Parallel Programming Using C++*, G. V. Wilson and P. Lu, Eds. Cambridge, MA: MIT Press, 1996, pp. 175–213.

[76] M. Bhandarkar, L. V. Kalé, E. de Sturler, and J. Hoeflinger, "Adaptive Load Balancing for MPI Programs," in *Proceedings of the International Conference on Computational Science (ICCS 2001)*, ser. Lecture Notes in Computer Science (LNCS), vol. 2074/2001. San Francisco, CA: Springer Verlag: Heidelberg, Germany, May 2001, pp. 108–117.

[77] L. V. Kalé, S. Kumar, and J. DeSouza, "A Malleable-Job System for Timeshared Parallel Machines," in *Proceedings of the 2nd International Symposium on Cluster Computing and the Grid (CCGrid 2002)*. Berlin, Germany: IEEE Computer Society Press: Los Alamitos, CA, May 2002, pp. 215–222.

[78] D. Abramson, R. Sosic, J. Giddy, and B. Hall, "Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations," in *Proceedings of the 4th International Symposium on High Performance Distributed Computing (HPDC4)*. Pentagon City, VA: IEEE Computer Society Press: Los Alamitos, CA, Aug. 1995, pp. 112–121.

[79] R. Buyya, J. Giddy, and D. Abramson, "An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications," in *Proceedings of the 2nd Annual Workshop on Active Middleware Services (AMS 2000)*. Pittsburgh, PA: Kluwer Academic Publishers: Dordrecht, Netherlands, Aug. 2000.